JOURNAL OF COMPUTER AND SYSTEM SCIENCES 8, 71–83 (1974)

Program Schemata and the First-Order Decision Problem*

HARRY R. LEWIS

Center for Research in Computing Technology, Harvard University, Cambridge, Massachusetts 02138

In [7] Manna reduces certain decision problems for program schemata to the problem of determining whether or not a first-order formula is satisfiable. Here we give an improved construction, which yields new positive results and suggests a method that may be useful in further investigations.

INTRODUCTION

We may distinguish two approaches to the problem of verifying assertions about formal computer programs. One class of methods includes a variety of inductive and fixed-point arguments based on the semantics of programs [8]; the other techniques, more syntactic and combinatorial, encode formal programming languages into more tractable formal systems. Manna [7] has shown that the language of first-order logic is adequate to represent a large class of formal programs and notes that this representation, combined with classical results on the decision problem for first-order logic (i.e., the problem of deciding whether or not a well-formed formula is satisfiable), yields the decidability of some properties for certain restricted types of programs. Our main result (see the end of §1 below) is that the first-order representation of Manna can be modified so as to yield further sufficient conditions for decidability.

The system of first-order logic used by Manna in [7] is the predicate calculus with function symbols but without identity; this system we denote by Q_f. The system Q_f is reducible to the predicate calculus without function symbols and without identity; this system we denote by Q. (In this context, "reducible" means that we can effectively find, for any formula F of Q_f, a formula F' of Q such that F is satisfiable iff F' is satisfiable; see [5, pp. 148–172].) For inquiries of the kind pursued in [7] and in this paper, the system Q has the advantage that its decision problem has been studied more thoroughly, and with more positive results, than that for Q_f [1, 3].

* This research has been supported in part by the National Science Foundation under Grant NSF-GJ-30409 and by the Center for Research in Computing Technology under the Division of Engineering and Applied Physics, Harvard University. The author held a National Science Foundation Traineeship during the period September 1971–August 1972.

However, an attempt to study the decision problem for the formulae of Q_f derived in [7] by reducing them to formulae of Q would be fruitless because of the complexity of the reduction. Instead, we return to the formal computer programs themselves and reduce directly from these programs to formulae of Q in such a way that the resulting formulae are simple enough to study in detail. In §3 we give an example of a formal computer program and the first-order formulae derived from it by these various reduction methods.

The main theorem (end of §1 below) states that the problems of totality and total-equivalence (defined in §0B) for program schemata can be reduced to the decision problem for Q. As a corollary (Corollary 1) to the proof of this theorem, these properties are shown to be decidable for *full* schemata (which were introduced in [10]). Also (Corollary 2), we define an analogue for program schemata to the property of *finite controllability* for first-order formulae, and show that for any class of program schemata possessing this property, the problems of totality and total-equivalence are decidable.

Preliminaries

A. First-order logic

We assume that the reader is familiar with the principles of first-order logic as presented, for example, in [5]; in this section we give a precise statement of one version of the Skolem–Herbrand–Gödel Theorem.[1]

The *functional form* F^* of a closed prenex formula[2] F of Q is formed by selecting, for each existentially quantified variable x, a distinct *indicial function symbol* f_x, replacing x by $f_x(y_1,\ldots,y_n)$ wherever x occurs in the matrix of F, where $y_1,\ldots,y_n$ are the universally quantified variables whose quantifiers lie to the left of $(\exists x)$, and deleting the quantifier prefix. The *Herbrand universe* $D(F)$ is the (usually infinite) set of all terms that can be constructed from the indicial function symbols appearing in F^* (and a 0-adic function symbol, if none appears in F^*). A *lexical instance* of F is the result of substituting terms $t_1,\ldots,t_n$ from $D(F)$ for the free variables $y_1,\ldots,y_n$ of F^*. The *Herbrand expansion* $\mathscr{H}(F)$ is the (usually infinite) set of all lexical instances of F. The *Skolem–Herbrand–Gödel Theorem* states that F is unsatisfiable iff some

[1] This theorem, which lies at the heart of the Completeness Theorem, is often mistakenly referred to as "Herbrand's Theorem," but that name should be reserved for the constructively proved result that: (a) from a formal refutation of F in a Hilbert-type system, we can primitive recursively construct an inconsistent set of lexical instance of F, and (b) from an inconsistent set of lexical instances of F, we can primitive recursively construct a (normal form) refutation of F. See [2, p. 45, footnote 3], [4], and the papers by Skolem, Herbrand, and Gödel in [11].

[2] The term "schema," which in the literature of formal logic is sometimes used to refer to well-formed formulae, is used in this paper only as an abbreviation for "program schema."

finite conjunction of lexical instances of F is (truth-functionally) inconsistent. We shall use this fundamental theorem in the form: a formula F in conjunctive normal form is satisfiable iff there is a truth-assignment $\mathcal{A}$ that verifies at least one disjunct in every clause (i.e., conjunct) of every lexical instance in $\mathcal{H}(F)$. (For "$\mathcal{A}$ verifies A," we write "$\mathcal{A} \models A$.")

B. Program schemata

For the most part we adopt the notion of "program schema" from [6]; the reader is referred to that paper for an explanation of the relation between program schemata and computer programs.

The following symbols and sets of symbols are used in the sequel:

(i) the set of numerals (we shall not distinguish between a numeral and the natural number for which it stands);

(ii) for each $i \geqslant 0$, an infinite set $\mathscr{F}_i$ of i-adic *function symbols*[3];

(iii) another infinite set $\mathscr{B}$ of 0-adic function symbols[4] (i.e., constant symbols);

(iv) an infinite set $\mathscr{P}$ of monadic *predicate symbols*[5];

(v) an infinite set $\mathscr{L}$ of *location symbols*;

(vi) the symbols ":=," "STOP," parentheses, and comma.

The sets $\mathscr{F}_i$, $\mathscr{B}$, $\mathscr{P}$, and $\mathscr{L}$ are pairwise disjoint. With each of $\mathscr{B}$, $\mathscr{L}$, and $\mathscr{F} = \bigcup_{i=0}^{\infty} \mathscr{F}_i$, associate some fixed ordering of its members, and let F_i, B_i, L_i denote the ith member of $\mathscr{F}$, $\mathscr{B}$, $\mathscr{L}$, respectively. Let F, P, L be syntactic variables ranging over $\mathscr{F}$, $\mathscr{P}$, $\mathscr{L}$, respectively.

A *program schema*, or simply *schema*, is a finite list $\langle I_1, \ldots, I_s \rangle$ of *instructions*, where each instruction I_k is of one of the following three forms:

(a) *Assignment instructions*

$$L := F(L_{i_1}, \ldots, L_{i_l}),$$

where $F \in \mathscr{F}_l$ ("Assign $F(L_{i_1}, \ldots, L_{i_l})$ to location L and proceed to instruction I_{k+1}").

(b) *Transfer instructions*

$$P(L)m, n,$$

where $m, n \leqslant s$ ("Go to instruction I_m if P is true of the current contents of location L, otherwise go to instruction I_n").

[3] Called "operator symbols" in [6].

[4] For which the location symbols are used in [6].

[5] Called "transfer symbols" in [6].

(c) *Stop instruction*

STOP,

("Stop").

Without loss of generality we assume that a program schema $S = \langle I_1, \ldots, I_s \rangle$ has exactly one stop instruction,[6] viz. I_{H_S}, that I_s is not an assignment instruction, and that if p location symbols appear in S, those symbols are $L_1, \ldots, L_p$.

Let $S = \langle I_1, \ldots, I_s \rangle$ be a program schema in which p location symbols appear. If r is the largest subscript on any function symbol appearing in S, then let $\mathscr{T}_S$ be the (usually infinite) set of all terms that can be formed from the constant symbols $B_1, \ldots, B_p$ and the function symbols $F_1, \ldots, F_r$. We use t, t_i, and t_i' $(i > 0)$ as variables ranging over $\mathscr{T}_S$. An *interpretation*[7] $\mathscr{I}$ for S is a mapping from the set of predicate symbols appearing in S to the set of characteristic functions with domain $\mathscr{T}_S$.

We now define the notion of a "computation" by a program schema. To do so we introduce concepts similar to those used in automata theory. At any point during a computation by a program schema, the status of the computation is described by an "instantaneous description," the components of which specify the instruction that is about to be "executed" and the "contents" of the locations. A complete computation is specified by a series of instantaneous descriptions: the first in the sequence is "initial," and each of the others stands in the relation of "immediate successor" to the preceding one in the sequence. Formally, let S be a program schema in which p location symbols appear, and let $\mathscr{T}_S{}^p$ be the p-fold cartesian product of $\mathscr{T}_S$ with itself. An *instantaneous description* of S is a member of $\{1, \ldots, s\} \times \mathscr{T}_S{}^p$, and the binary relation $\vdash_{S,\mathscr{I}}$ ("yields in one step") between instantaneous descriptions is defined to be the smallest relation satisfying (a) and (b):

(a) If I_k is $L_j := F(L_{i_1}, \ldots, L_{i_l})$, then $(k, t_1, \ldots, t_p) \vdash_{S,\mathscr{I}} (k+1, t_1, \ldots, t_{j-1}, F(t_{i_1}, \ldots, t_{i_l}), t_{j+1}, \ldots, t_p)$.

(b) If I_k is $P(L_j)m, n$, then $(k, t_1, \ldots, t_p) \vdash_{S,\mathscr{I}} (m, t_1, \ldots, t_p)$ if $\mathscr{I}(P)(t_j) = 1$, and $(k, t_1, \ldots, t_p) \vdash_{S,\mathscr{I}} (n, t_1, \ldots, t_p)$ if $\mathscr{I}(P)(t_j) = 0$.

Let $\vdash^*_{S,\mathscr{I}}$ be the reflexive, transitive closure of $\vdash_{S,\mathscr{I}}$. *S computes the value* $\langle t_1, \ldots, t_p \rangle$ *under* $\mathscr{I}$ if $(1, B_1, \ldots, B_p) \vdash^*_{S,\mathscr{I}} (H_S, t_1, \ldots, t_p)$; *$S$ diverges* under $\mathscr{I}$ if S computes no value under $\mathscr{I}$. An easy theorem is that program schemata are deter-

[6] If more than one of $I_1, \ldots, I_s$ is a stop instruction, then let I_{H_S} be one of them, and replace the others by the instruction $P(L_1)H_S, H_S$, where P is any member of $\mathscr{P}$. If none of $I_1, \ldots, I_s$ is a stop instruction, then the schema is trivial for our purposes: it diverges under all interpretations.

[7] In [6] and [7], an "interpretation" also includes assignments of elements from some set D to the constant symbols of $\mathscr{T}_S$ and functions on D to the function symbols from which $\mathscr{T}_S$ is constructed. Our definition is based on the fact that for the investigations carried out in this paper we may take D to be $\mathscr{T}_S$.

ministic, i.e., under a given interpretation, a schema computes at most one value. A program schema S is *total*[8] if S computes a value under every interpretation for S. Two schemata are *total-equivalent*[9] if each is total and each computes the same value as the other under all common interpretations.

In §1 and 2 we shall derive criteria for totality and total-equivalence of program schemata in terms of the satisfiability of certain first-order formulae; in special cases, this criterion yields a decision procedure for these properties. We now sketch an argument to the effect that total-equivalence is likely to be decidable for a class $\mathscr{C}$ of schemata if totality is decidable for $\mathscr{C}$, provided that the specification of $\mathscr{C}$ satisfies certain general properties.

Suppose that totality is decidable for $\mathscr{C}$ and we wish to determine whether two given schemata S_1 and S_2 are total-equivalent. Assume that S_1 and S_2 each have p locations. The idea is to build a third schema S_3 with $2p$ locations, to simulate the computations of S_1 and S_2 separately, and then to compare the results. S_3 can be constructed so as to diverge just in case S_1 or S_2 diverges or they compute different values, so that S_3 is total iff S_1 and S_2 are total-equivalent. Thus, a decision procedure for totality for $\mathscr{C}$ can be used to decide total-equivalence, if the constructed schema S_3 is in $\mathscr{C}$ for any S_1, S_2 in $\mathscr{C}$.

§1. The Construction

We now proceed to the construction of first-order formulae with neither function symbols nor the identity sign expressing the computations of program schemata.

Let $S = \langle I_1,\ldots, I_s\rangle$ be a program schema in which p location symbols appear. Let K be the set of all numbers k such that I_k is an assignment instruction. For each $k \in K$, if I_k is $L := F(L_{i_1},\ldots, L_{i_l})$, then let $\lambda_k = l$, and for $j = 1,\ldots, l$, let $\pi_k(j) = i_j$. Let η_k be the cardinality of the set $\{1,\ldots, p\} - E_k$, where $E_k = \{i_1,\ldots, i_l\}$. Thus, λ_k is the number of argument places of the function symbol appearing in I_k, and η_k is the number of location symbols not among the arguments of that function symbol in I_k. Let $\lambda = \max\{\lambda_k \mid k \in K\}$ and $\eta = \max\{\eta_k \mid k \in K\}$. For each $k \in K$, extend the function π_k to be onto the set $\{1,\ldots, p\}$ by setting $\pi_k(\lambda_k + i) = l_i$ for $i = 1,\ldots, \eta_k$, where $l_1,\ldots, l_{\eta_k}$ is an exhaustive and irredundant listing of the members of $\{1,\ldots, p\} - E_k$. Let τ_k be any inverse of π_k, and let $\gamma = \max\{p, \lambda\}$, $q = \lambda + \eta + \gamma - p$.

Let x_i, y_j, z_k, u_m be individual variables ($i \geqslant 0$; $j, k, m > 0$), and let Q_i, Q_i' ($i > 0$) be new q-adic predicate letters. Let M_S be the conjunction of all the clauses from (1)–(4) below:

[8] In [7], S is said to "terminate."

[9] In [7], the schemata are said to be "equivalent" (with respect to the null predicate).

(1) $(Q_1 x_1 \cdots x_p x_0 \cdots x_0)$.

(2) For each k such that I_k is a transfer instruction[10] $P(L_j)m, n$,

$$(\neg Q_k y_1 \cdots y_\nu x_0 \cdots x_0 \vee \neg P y_j \vee Q_m y_1 \cdots y_\nu x_0 \cdots x_0)$$
$$\& (\neg Q_k y_1 \cdots y_\nu x_0 \cdots x_0 \vee P y_j \vee Q_n y_1 \cdots y_\nu x_0 \cdots x_0).$$

(3) For each $k \in K$,

$$(\neg Q_k y_1 \cdots y_\nu x_0 \cdots x_0$$
$$\vee Q_k' y_{\pi_k(1)} \cdots y_{\pi_k(\lambda_k)} x_0 \cdots x_0 y_{\pi_k(\lambda_k+1)} \cdots y_{\pi_k(\lambda_k+\eta_k)} y_{p+1} \cdots y_\lambda x_0 \cdots x_0),$$

where in the second disjunct "$y_{\pi_k(\lambda_k+1)}$" is at argument position $\lambda + 1$. (The string "$y_{p+1} \cdots y_\lambda$" is empty if $\lambda \leqslant p$.)

(4) For each $k \in K$, if I_k is $L_m := F_n(L_{i_1}, \ldots, L_{i_l})$,

$$(\neg Q_k' y_1 \cdots y_\lambda u_1 \cdots u_{\eta_k} x_0 \cdots x_0 \vee Q_{k+1} v_1 \cdots v_{m-1} z_n v_{m+1} \cdots v_p x_0 \cdots x_0),$$

where

$$v_i \text{ is } \begin{cases} y_{\tau_k(i)} & \text{if } i \in E_k \\ u_{\tau_k(i)-\lambda_k} & \text{if } i \in \{1, \ldots, p\} - E_k . \end{cases}$$

Now let r be the largest subscript on any function symbol appearing in S, and let W_S be the formula of Q

$$(\exists x_0) \cdots (\exists x_p)(y_1) \cdots (y_\lambda)(\exists z_1) \cdots (\exists z_r)(y_{\lambda+1}) \cdots (y_p)(u_1) \cdots (u_\eta)$$
$$[M_S \& \neg Q_{H_S} y_1 \cdots y_\nu x_0 \cdots x_0].$$

Let T be another program schema in which p location symbols appear, and without loss of generality assume that the parameters λ, r, η are the same for T as for S.[11] Let $\mathbf{M}_T$ be the matrix derived from M_T by making boldface[12] the predicate letters Q_i, Q_i', and let $W_{S,T}$ be the formula of Q with the same prefix as W_S and with matrix

$$[M_S \& \mathbf{M}_T \& (\neg Q_{H_S} y_1 \cdots y_\nu x_0 \cdots x_0 \vee \neg \mathbf{Q}_{H_T} y_1 \cdots y_\nu x_0 \cdots x_0)].$$

Finally, we can state the theorem.

[10] We use the *predicate symbols* appearing in program schemata also as monadic *predicate letters* in our first-order language.

[11] One way of bringing about this situation is as follows. Let $S = \langle I_1, \ldots, I_s \rangle$ and $T = \langle J_1, \ldots, J_{s'} \rangle$. Then instead of using S and T in the construction, use $S' = \langle I_1, \ldots, I_s, J_1, \ldots, J_{s'} \rangle$ and $T' = \langle J_1, \ldots, J_{s'}, I_1, \ldots, I_s \rangle$.

[12] A boldface predicate letter $\mathbf{Q}_i$ is to be construed as a new predicate letter with the same number of argument places as Q_i.

THEOREM. (a) *The first-order formula W_S is unsatisfiable iff the program schema S is total.*

(b) *The first-order formula $W_{S,T}$ is unsatisfiable iff the program schemata S and T are total-equivalent.*

§2. PROOF OF THE THEOREM

We first set up mappings between the Herbrand universe $D(W_S)$ (which is the same as $D(W_{S,T})$) and the set $\mathscr{T}_S$. Let $a, b_1, \ldots, b_p$ be the 0-adic indicial function symbols replacing $x_0, \ldots, x_p$, respectively, in the functional form $W_S{}^*$, and let $f_1, \ldots, f_r$ be the λ-adic indicial function symbols replacing $z_1, \ldots, z_r$. Then define mappings ϕ: $D(W_S) \to \mathscr{T}_S$ and ψ: $\mathscr{T}_S \to D(W_S)$ recursively as follows:

$$\phi(a) = B_1,$$
$$\phi(b_i) = B_i \qquad (i = 1, \ldots, p),$$

for any $t_1, \ldots, t_\lambda \in D(W_S)$, and any $i \leqslant r$ such that $F_i \in \mathscr{F}_j$,

$$\phi(f_i(t_1, \ldots, t_\lambda)) = F_i(\phi(t_1), \ldots, \phi(t_j)),$$
$$\psi(B_i) = b_i \qquad (i = 1, \ldots, p),$$

for any $F_i \in \mathscr{F}_j$ and any $t_1, \ldots, t_j \in \mathscr{T}_S$,

$$\psi(F_i(t_1, \ldots, t_j)) = f_i(\psi(t_1), \ldots, \psi(t_j), a, \ldots, a).$$

The theorem follows from four lemmas. Part (a) of the theorem follows from Lemmas 1 and 2, part (b) from Lemmas 3 and 4.

LEMMA 1. *If S is total, then W_S is unsatisfiable.*

Proof. Assume to the contrary that S is total but W_S is satisfiable (we deduce a contradiction from this assumption). By the Skolem–Herbrand–Gödel Theorem, there is a truth-assignment $\mathscr{A}$ verifying the Herbrand expansion $\mathscr{H}(W_S)$. Let $\mathscr{I}$ be the interpretation for S under which $\mathscr{I}(P)(t) = 1$ iff $\mathscr{A} \models P\psi(t)$.

SUBLEMMA. *If*

$$(1, B_1, \ldots, B_p) \vdash^*_{S,\mathscr{I}} (k, t_1, \ldots, t_p),$$

then $\mathscr{A} \models Q_k\psi(t_1) \cdots \psi(t_p)a \cdots a$.

Proof of Sublemma. (i) The relation $\vdash^*_{S,\mathscr{I}}$ is reflexive, so

$$(1, B_1, \ldots, B_p) \vdash^*_{S,\mathscr{I}} (1, B_1, \ldots, B_p).$$

Also, the functional form of clause (1) is just $Q_1\psi(B_1)\cdots\psi(B_p)a\cdots a$, so the sublemma holds at the beginning of the computation by S.

Now suppose $(1, B_1,..., B_p) \vdash^*_{S,\mathscr{I}} (k, t_1,..., t_p) \vdash_{S,\mathscr{I}} (k', t_1',..., t_p')$, and assume as the inductive hypothesis that $\mathscr{A} \models Q_k\psi(t_1)\cdots\psi(t_p)a\cdots a$.

(ii) If I_k is $P(L_j)m, n$ and $\mathscr{I}(P)(t_j) = 1$, then $t_i = t_i'$ for each i, $k' = m$, and $\mathscr{A} \models P\psi(t_j)$. Let Z be any lexical instance in which $\psi(t_1),..., \psi(t_p)$ are substituted for $y_1,..., y_p$, respectively, and a is substituted for $y_{p+1},..., y_\gamma$. Then, by (2) of the construction, Z contains the clause

$$(\neg Q_k\psi(t_1)\cdots\psi(t_p)a\cdots a \vee \neg P\psi(t_j) \vee Q_m\psi(t_1)\cdots\psi(t_p)a\cdots a).$$

Since $\mathscr{A}$ falsifies the first two disjuncts but verifies $\mathscr{H}(W_S)$, $\mathscr{A} \models Q_m\psi(t_1)\cdots\psi(t_p)a\cdots a$, which is the same as $Q_{k'}\psi(t_1')\cdots\psi(t_p')a\cdots a$. The case in which $\mathscr{A} \models \neg P\psi(t_j)$ follows similarly.

(iii) If I_k is the assignment instruction $L_m := F_n(L_{\pi_k(1)},..., L_{\pi_k(\lambda_k)})$, then $t_i' = t_i$ for $i \neq m$, and $t_m = F_n(t_{\pi_k(1)},..., t_{\pi_k(\lambda_k)})$. Let us say that $F_n(t_{\pi_k(1)},..., t_{\pi_k(\lambda_k)})$ is the *function call*; first, we explain informally how the clauses (3) and (4) are used to "simulate" this function call. Argument positions $\lambda + 1,..., q$ of the predicate letter Q_k' may be regarded as "storage" positions; those terms among $t_1,..., t_p$ that are not to be among the arguments of the function call are "moved" into these positions by clause (3), and, simultaneously, the terms that are to be in the argument list are distributed in positions $1,..., \lambda$, with the "dummy" constant a filling unused positions. In clause (4), the function is "applied" to the terms in positions $1,..., \lambda$ of Q_k', the result is put into position m of Q_{k+1}, and the terms among $t_1,..., t_p$ that were not changed by this instruction are restored in their original positions.

We now return to the formal proof of the sublemma. By (3) of the construction and by examination of the lexical instance Z used in (ii), it follows that

$$\mathscr{A} \models Q_k'\psi(t_{\pi_k(1)})\cdots\psi(t_{\pi_k(\lambda_k)})\,a\cdots a\psi(t_{\pi_k(\lambda_k+1)})\cdots\psi(t_{\pi_k(\lambda_k+\eta_k)})\,a\cdots a,$$

where "$\psi(t_{\pi_k(\lambda_k+1)})$" is at argument position $\lambda + 1$. Then consider a lexical instance in which the following substitutions are made:

$$\begin{aligned} \psi(t_{\pi_k(i)}) \quad &\text{for} \quad y_i \quad (i = 1,..., \lambda_k), \\ a \quad &\text{for} \quad y_i \quad (i = \lambda_k + 1,..., \lambda), \\ \psi(t_{\pi_k(\lambda_k+i)}) \quad &\text{for} \quad u_i \quad (i = 1,..., \eta_k). \end{aligned}$$

By (4) of the construction and the facts that τ_k is an inverse of π_k and that z_n is replaced in $W_S{}^*$ by $f_n(y_1,..., y_\lambda)$, it follows that

$$\mathscr{A} \models Q_{k+1}\psi(t_1)\cdots\psi(t_{m-1})f_n(\psi(t_{\pi_k(1)}),..., \psi(t_{\pi_k(\lambda_k)}), a,..., a)\,\psi(t_{m+1})\cdots\psi(t_p)\,a\cdots a,$$

which is the same as $Q_{k'}\psi(t_1')\cdots\psi(t_p')a\cdots a$. ∎

Now back to the proof of Lemma 1.

Since S is total, there are terms $t_1, \ldots, t_p \in D(W_S)$ such that $\mathcal{A} \models Q_{H_S} t_1 \cdots t_p a \cdots a$. Then $\mathcal{A}$ falsifies an instance of $Q_{H_S} y_1 \cdots y_\nu x_0 \cdots x_0$, which is a clause of W_S, and does not verify $\mathcal{H}(W_S)$, contrary to the original assumption. ▮

LEMMA 2. *If S is not total, then W_S is satisfiable.*

Proof. Let $\mathcal{I}$ be an interpretation under which S diverges. Define a truth-assignment $\mathcal{A}$ on $\mathcal{H}(W_S)$ as follows:

For predicate symbols P appearing in S, $\mathcal{A} \models Pt$ iff $\mathcal{I}(P)(\phi(t)) = 1$.

For the predicate letters Q_k: $\mathcal{A} \models Q_k t_1 \cdots t_q$ iff $(1, B_1, \ldots, B_p) \overset{*}{\vdash}_{S,\mathcal{I}} (k, \phi(t_1), \ldots, \phi(t_p))$ and $t_{p+1} = t_{p+2} = \cdots = t_q = a$.

For the predicate letters Q_k': $\mathcal{A} \models Q_k' t_1 \cdots t_q$ iff $t_{\lambda_k+1} = \cdots = t_\lambda = t_{\lambda+\eta_k+1} = \cdots = t_q = a$, and there are terms $t_1', \ldots, t_p' \in \mathcal{T}_S$ such that $(1, B_1, \ldots, B_p) \overset{*}{\vdash}_{S,\mathcal{I}} (k, t_1', \ldots, t_p')$ and

(i) for $i = 1, \ldots, \lambda_k$, $\phi(t_i) = t'_{\pi_k(i)}$;

(ii) for $i = 1, \ldots, \eta_k$, $\phi(t_{\lambda+i}) = t'_{\pi_k(\lambda_k+i)}$.

A case-analysis similar to that in the proof of Lemma 1 establishes that $\mathcal{A}$ verifies at least one disjunct in each clause of every lexical instance of W_S, and thus $\mathcal{A} \models \mathcal{H}(W_S)$. ▮

LEMMA 3. *If S and T are total-equivalent, then $W_{S,T}$ is unsatisfiable.*

Proof. Suppose S and T are total-equivalent, but $W_{S,T}$ is satisfiable. (As before, we derive a contradiction.) Let $\mathcal{A}$ be a truth-assignment verifying $\mathcal{H}(W_{S,T})$. As in the proof of Lemma 1, let $\mathcal{I}$ be the interpretation determined by $\mathcal{A}$. Then by the sublemma of Lemma 1, if $\langle t_1, \ldots, t_p \rangle$ is the value computed by S and T under $\mathcal{I}$, then

$$\mathcal{A} \models Q_{H_S} \psi(t_1) \cdots \psi(t_p)\, a \cdots a \;\&\; \mathbf{Q}_{H_T} \psi(t_1) \cdots \psi(t_p)\, a \cdots a.$$

But then $\mathcal{A}$ falsifies a clause of $\mathcal{H}(W_{S,T})$ and does not verify $\mathcal{H}(W_{S,T})$, contrary to the original assumption. ▮

LEMMA 4. *If S and T are not total-equivalent, then $W_{S,T}$ is satisfiable.*

Proof. Let $\mathcal{I}$ be an interpretation under which S or T diverges or under which S and T compute different values. Construct a truth-assignment $\mathcal{A}$ on $\mathcal{H}(W_{S,T})$ from $\mathcal{I}$ as in the proof of Lemma 2, using the relation $\overset{*}{\vdash}_{S,\mathcal{I}}$ to determine the values of $\mathcal{A}$ on instances of Q_i, Q_i', and $\overset{*}{\vdash}_{T,\mathcal{I}}$ to determine $\mathcal{A}$ on instances of $\mathbf{Q}_i$, $\mathbf{Q}_i'$. Then $\mathcal{A}$ verifies all instances of M_S and $\mathbf{M}_T$; moreover, since S or T diverges under $\mathcal{I}$, or they compute different values under $\mathcal{I}$, $\mathcal{A}$ verifies each instance of

$$(\neg Q_{H_S} y_1 \cdots y_\nu x_0 \cdots x_0 \vee \neg \mathbf{Q}_{H_T} y_1 \cdots y_\nu x_0 \cdots x_0). \;▮$$

Here we give an example of the construction presented in §1 and contrast it with Manna's construction [7]. The example is not intended to show the advantages of our method over Manna's; it is hoped that the corollaries will justify the complex construction in §1.

Let S be the program schema $\langle I_1, \ldots, I_6 \rangle$, where

$$
\begin{aligned}
I_1 &= L_2 := F_1(L_2), \\
I_2 &= P(L_1)\ 5,\ 5, \\
I_3 &= L_2 := F_2(L_1, L_2), \\
I_4 &= L_1 := F_3(L_1), \\
I_5 &= P(L_1)\ 6,\ 3, \\
I_6 &= \text{STOP}
\end{aligned}
$$

(S is the program schema used as an example in [6, p. 222].)

(a) The result[13] of applying Manna's algorithm [7] to S is the formula W_1 of Q_f:

$$
\begin{aligned}
&(y_1)(y_2)[(Q_1 b_1 b_2) \\
&\quad \& (\neg Q_1 y_1 y_2 \vee Q_2 y_1 f_1(y_2)) \\
&\quad \& (\neg Q_2 y_1 y_2 \vee \neg P y_1 \vee Q_5 y_1 y_2) \\
&\quad \& (\neg Q_2 y_1 y_2 \vee P y_1 \vee Q_5 y_1 y_2) \\
&\quad \& (\neg Q_3 y_1 y_2 \vee Q_4 y_1 f_2(y_1, y_2)) \\
&\quad \& (\neg Q_4 y_1 y_2 \vee Q_5 f_3(y_1) y_2) \\
&\quad \& (\neg Q_5 y_1 y_2 \vee \neg P y_1 \vee Q_6 y_1 y_2) \\
&\quad \& (\neg Q_5 y_1 y_2 \vee P y_1 \vee Q_3 y_1 y_2) \\
&\quad \& (\neg Q_6 y_1 y_2)].
\end{aligned}
$$

(b) One algorithm for eliminating terms from formulae of Q_f reduces W_1 to the formula W_2 of Q by introducing new predicate letters to represent the relation between the value of a function and its arguments:

$$
\begin{aligned}
&(\exists x_1)\, X_1 x_1 \,\&\, (\exists x_2)\, X_2 x_2 \,\&\, (z_1)(\exists x_3)\, X_3 z_1 x_3 \\
&\quad \&\, (z_2)(z_3)(\exists x_4)\, X_4 z_2 z_3 x_4 \,\&\, (z_4)(\exists x_5)\, X_5 z_4 x_5 \\
&\quad \&\, (y_1)(y_2)(y_3)(y_4)(y_5)(y_6)(y_7)
\end{aligned}
$$

[13] Manna does not use the same notion of "program schema" as is used in [6] and here; we give the formula (somewhat simplified) constructed by his algorithm for the "abstract program" corresponding to S.

$$
\begin{aligned}
&[\neg X_1 y_3 \vee \neg X_2 y_4 \vee \neg X_3 y_2 y_5 \vee \neg X_4 y_1 y_2 y_6 \vee \neg X_5 y_1 y_7 \\
&\vee (Q_1 y_3 y_4) \\
&\quad \& (\neg Q_1 y_1 y_2 \vee Q_2 y_1 y_5) \\
&\quad \& (\neg Q_2 y_1 y_2 \vee \neg P y_1 \vee Q_5 y_1 y_2) \\
&\quad \& (\neg Q_2 y_1 y_2 \vee P y_1 \vee Q_5 y_1 y_2) \\
&\quad \& (\neg Q_3 y_1 y_2 \vee Q_4 y_1 y_6) \\
&\quad \& (\neg Q_4 y_1 y_2 \vee Q_5 y_7 y_2) \\
&\quad \& (\neg Q_5 y_1 y_2 \vee \neg P y_1 \vee Q_6 y_1 y_2) \\
&\quad \& (\neg Q_5 y_1 y_2 \vee P y_1 \vee Q_3 y_1 y_2) \\
&\quad \& (\neg Q_6 y_1 y_2)].
\end{aligned}
$$

(c) Finally, we carry out the construction presented in §1. The parameters p, λ, η are respectively 2, 2, 1; hence $\gamma = 2$ and $q = 3$. Then W_S is

$$
\begin{aligned}
&(\exists x_0)(\exists x_1)(\exists x_2)(y_1)(y_2)(\exists z_1)(\exists z_2)(\exists z_3)(u_1) \\
&[(Q_1 x_1 x_2 x_0) \\
&\quad \& (\neg Q_1 y_1 y_2 x_0 \vee Q_1' y_2 x_0 y_1) \\
&\quad \& (\neg Q_1' y_1 y_2 u_1 \vee Q_2 u_1 z_1 x_0) \\
&\quad \& (\neg Q_2 y_1 y_2 x_0 \vee \neg P y_2 \vee Q_5 y_1 y_2 x_0) \\
&\quad \& (\neg Q_2 y_1 y_2 x_0 \vee P y_2 \vee Q_5 y_1 y_2 x_0) \\
&\quad \& (\neg Q_3 y_1 y_2 x_0 \vee Q_3' y_1 y_2 x_0) \\
&\quad \& (\neg Q_3' y_1 y_2 x_0 \vee Q_4 y_1 z_2 x_0) \\
&\quad \& (\neg Q_4 y_1 y_2 x_0 \vee Q_4' y_1 x_0 y_2) \\
&\quad \& (\neg Q_4' y_1 y_2 u_1 \vee Q_5 z_3 u_1 x_0) \\
&\quad \& (\neg Q_5 y_1 y_2 x_0 \vee \neg P y_1 \vee Q_6 y_1 y_2 x_0) \\
&\quad \& (\neg Q_5 y_1 y_2 x_0 \vee P y_1 \vee Q_3 y_1 y_2 x_0) \\
&\quad \& (\neg Q_6 y_1 y_2 x_0)].
\end{aligned}
$$

§4. Corollaries

Let S be a program schema in which p location symbols appear. The schema S is said to be *full* iff for every assignment instruction $L := F(L_{i_1},\ldots,L_{i_l})$ appearing in S, $\{i_1,\ldots,i_l\} = \{1,\ldots,p\}$. Full schemata are studied in [10]; there Paterson shows that strong equivalence[14] of full schemata is decidable. We have a similar result:

[14] Two program schemata S and T are *strongly equivalent* iff under any interpretation $\mathcal{I}$ for S and T, either S and T both diverge, or S and T compute the same value. See [6, 10].

Proof. We rely on the following result concerning the decision problem for Q [3, 9]:

Let $\mathscr{S}$ be the class of prenex formulae that contains a formula F iff the prefix of F is of the form

$$(\exists x_0) \cdots (\exists x_a)(y_1) \cdots (y_b)(\exists z_1) \cdots (\exists z_c),$$

and the argument set of every atomic formula in the matrix of F includes either one of $z_1, \ldots, z_c$ or all of $y_1, \ldots, y_b$, or else at most one of $y_1, \ldots, y_b$. Then the decision problem is solvable for $\mathscr{S}$. (The class $\mathscr{S}$ is called the *extended Skolem class* in [3].)

Now if S and T are full schemata, then in the construction in §1 $\eta = 0$, $\lambda \geqslant p$, and $\gamma = \lambda$. Then the prefixes of W_S and $W_{S,T}$ are in the form just stated, and the reader may verify that the restrictions on atomic formulae are satisfied as well. ∎

A class $\mathscr{C}$ of first-order formulae is said to be *finitely controllable* if every satisfiable formula in $\mathscr{C}$ has a finite model. Any finitely controllable class of formulae has a solvable decision problem, since the class of unsatisfiable formulae and the class of formulae with finite models are both recursively enumerable. A model for a first-order formula induces a partition on its Herbrand universe: all terms standing for the same element of the model fall in the same partition class. Similarly, an interpretation $\mathscr{I}$ for a program schema S is called *finite* if, under $\mathscr{I}$, the terms in $\mathscr{T}_S$ may be regarded as names for a finite number of objects. (For a precise explanation, see [6].) Moreover, let us say that a class $\mathscr{C}$ of program schemata is *finitely divergent* if any program schema in $\mathscr{C}$ that is not total diverges under a finite interpretation.[15] The next corollary is easily proved by exploiting the natural correlation between the set $\mathscr{T}_S$ and the Herbrand universe $D(W_S)$.

COROLLARY 2. *Let $\mathscr{C}$ be a class of program schemata. Then*

(a) *$\mathscr{C}$ is a finitely divergent class of program schemata iff the class $\{W_S \mid S \in \mathscr{C}\}$ is a finitely controllable class of first-order formulae, and*

(b) *if $\mathscr{C}$ is a finitely divergent class of program schemata, then totality and total-equivalence are decidable for $\mathscr{C}$.*

Corollary 2(b) can also be proved directly by an argument like that of [6, Theorem 4.2]. For under a given finite interpretation, a program schema is no more than a finite automaton, so the relevant properties can be decided by well-known techniques. Then to tell, for example, whether a schema S from a finitely divergent class is total, we can systematically search for a finite interpretation under which S diverges at the same time as we are searching for a refutation of the formula W_S; we are assured that one of the searches will eventually be successful.

[15] An elegant example is given in [6, p. 227] of a program schema that computes a value under every finite interpretation but diverges under an infinite interpretation.

Now since the class $\mathscr{S}$ of extended Skolem formulae used in the proof of Corollary 1 is finitely controllable,[16] by Corollary 2 we have:

COROLLARY 3. *The class of full schemata is a finitely divergent class.*

This fact is not unexpected, as Paterson [10] exploits a similarity between finite automata and the *progressive* schemata, of which the full schemata are a subclass. But since finite divergence implies that totality is decidable, it may prove fruitful to focus directly on this property. What tools are available for showing that a class of program schemata contains only schemata that diverge, if at all, under a finite interpretation?

ACKNOWLEDGMENT

I am grateful to Ronald Book and Burton Dreben for their many helpful suggestions.

REFERENCES

1. W. ACKERMANN, "Solvable Cases of the Decision Problem," North-Holland, Amsterdam, 1954.
2. B. DREBEN, Solvable Suranyi subclasses: An introduction to the Herbrand theory, *Annals of the Harvard Computation Laboratory* **31** (1962), 32–47.
3. B. DREBEN AND W. D. GOLDFARB, A systematic treatment of the decision problem (forthcoming).
4. "Jacques Herbrand: Logical Writings" (W. D. Goldfarb, Ed.), Harvard University Press, Cambridge, MA, 1971.
5. S. KLEENE, "Mathematical Logic," John Wiley and Sons, New York, 1967.
6. D. LUCKHAM, D. PARK, AND M. PATERSON, On formalized computer programs, *J. Comput. System Sci.* **4** (1970), 220–249.
7. Z. MANNA, Properties of programs and the first-order predicate calculus, *J. Assoc. Comput. Mach.* **16** (1969), 244–255.
8. Z. MANNA, S. NESS, AND J. VUILLEMIN, Inductive methods for proving properties of programs, "Proceedings of an ACM Conference on Proving Assertions about Programs," 1972.
9. S. MASLOV, Application of the inverse method for establishing deducibility to the theory of decidable fragments in the classical predicate calculus, *Soviet Math. Dokl.* **7** (1966), 1653.
10. M. PATERSON, Equivalence problems in a model of computation, Doctoral Dissertation, Cambridge University, 1967; reissued as MIT Artificial Intelligence Lab. Memo #1, 1970.
11. J. VAN HEIJENOORT, Ed., "From Frege to Gödel: A Sourcebook in Mathematical Logic 1879–1931," Harvard University Press, Cambridge, MA, 1967.

[16] This follows from an argument similar to that of [2, p. 47, footnote 30]; the full proof is in [3]. In this case, as in general, the proof of finite controllability is much more difficult than the proof of solvability.